Extracting a Domain Specific Language From an Example

A Bottom-Up Method Using the *ngrease* Metalanguage

Ville T Oikarinen

Sysart Oy

ville@oikarinen.org

Abstract

This demonstration shows a lightweight and fast method for creating a tested and working domain specific language. The method is demonstrated using the *ngrease* metalanguage.

The creation of a new language is started by writing a representative example of the final product with a test that tests the transformation from a stub source to the result.

The test is made to pass by writing a constant transformer that unconditionally outputs the result.

At each step the language is extended by *refactoring*: Some part of the transformer template is converted from a constant subtree to a reference to data read from the source tree, thus driving additions to the new language.

Optionally, each refactoring step can be driven by a new test that demonstrates the lack of parameterization of some part of the final product.

Categories and Subject Descriptors D.2.3 [*Coding Tools and Techniques*]

General Terms Languages

Keywords metaprogramming, code generation, DSL, method, refactoring

1. The Problem

A common problem in software development is that the chosen programming languages and tools require too much "boilerplate" code: a set of closely related programming problems each require a unique solution.

Code generation helps, but the experience of the author is that it is not used often enough and well enough. One possible reason is that the practical problems and difficulties are (perceived to be) too big.

2. A Solution: Stepwise Refactoring Into a New Language

This demonstration shows a lightweight method for incrementally *refactoring* a concrete solution into a new domain specific language and a transformer that consumes code written in the new language and produces solutions similar to the original example.

2.1 A Tool: *ngrease*

The demonstration uses the *ngrease* language as the metalanguage so it is recommended to read its introduction [1] first. The method has its uses with traditional template languages, too.

ngrease is a pure functional template language that processes string trees unlike traditional template languages that produce raw string content directly.

2.2 Preliminary Step: Convert the Example To an AST

Since *ngrease* processes trees, it is easier to extract the DSL from an "ngreased" version of the (programming) language used in the example, an Abstract Syntax Tree -like tree model, instead of the actual raw string source code of the example.

In this demonstration the final product has already been translated to an *ngrease* tree, and a working transformer for it is given.

2.3 Starting Point: Test And Create a Constant Transformer

A language is defined by a transformer that consumes code written in the language.

Since the new language will be created by refactoring, we first need a working transformer that produces the given example.

This is achieved by creating a test that tests the transformer by feeding it a *stub* of a source in the new DSL and asserting that it produces the given example.

The test is made green by writing a constant transformer i.e. a transformer defined by the given example.

2.4 Optional: Test Missing Parameterization

After the first step the stub language passes all tests but is unusable. Ideally a new test is needed to demonstrate the need for some new parameter for creating different solutions.

In practice, however, it may be more difficult to write the test, especially the expected output, in advance than to “pin” the manually reviewed behaviour afterwards.

2.5 Refactoring: Parameterize a Subtree

This is the step that incrementally makes the DSL more general. The step is analogous to the Parameterize Method refactoring [2]: In the transformer definition, a subtree or a list of “similar” subtrees is replaced by a reference. Defining similarity here is part of the process.

The reference may point to data read directly from the source tree written in the DSL.

More often, however, the abstraction of the subtree is not suitable for the DSL so further transformations are needed for parts of it. This is a subproblem of the original so it can be solved by extracting a sub-DSL.

After this step, the tests are red until the sources written in the DSL are updated to the new, extended version of the DSL under construction.

3. Biography

Ville Oikarinen is a Finnish software developer who also occasionally works as an instructor. His main interests are laziness (automatization and software configuration management), expressiveness (languages and metaprogramming), freedom (portable and open software) and minimalism (). He is also the author of the *ngrease* language.

References

[1] `http://ngrease.sourceforge.net/introduction.html`

[2] `http://www.refactoring.com/catalog/parameterizeMethod.html`; accessed 25.7.2007